Squad dataset link : https://rajpurkar.github.io/SQuAD-explorer/

"""Official evaluation script for SQuAD version 2.0.

In addition to basic functionality, we also compute additional statistics and

plot precision-recall curves if an additional na\_prob.json file is provided.

This file is expected to map question ID's to the model's predicted probability

that a question is unanswerable.

"""

import argparse

import collections

import json

import numpy as np

import os

import re

import string

import sys

OPTS = None

def parse\_args():

parser = argparse.ArgumentParser('Official evaluation script for SQuAD version 2.0.')

parser.add\_argument('data\_file', metavar='data.json', help='Input data JSON file.')

parser.add\_argument('pred\_file', metavar='pred.json', help='Model predictions.')

parser.add\_argument('--out-file', '-o', metavar='eval.json',

help='Write accuracy metrics to file (default is stdout).')

parser.add\_argument('--na-prob-file', '-n', metavar='na\_prob.json',

help='Model estimates of probability of no answer.')

parser.add\_argument('--na-prob-thresh', '-t', type=float, default=1.0,

help='Predict "" if no-answer probability exceeds this (default = 1.0).')

parser.add\_argument('--out-image-dir', '-p', metavar='out\_images', default=None,

help='Save precision-recall curves to directory.')

parser.add\_argument('--verbose', '-v', action='store\_true')

if len(sys.argv) == 1:

parser.print\_help()

sys.exit(1)

return parser.parse\_args()

def make\_qid\_to\_has\_ans(dataset):

qid\_to\_has\_ans = {}

for article in dataset:

for p in article['paragraphs']:

for qa in p['qas']:

qid\_to\_has\_ans[qa['id']] = bool(qa['answers'])

return qid\_to\_has\_ans

def normalize\_answer(s):

"""Lower text and remove punctuation, articles and extra whitespace."""

def remove\_articles(text):

regex = re.compile(r'\b(a|an|the)\b', re.UNICODE)

return re.sub(regex, ' ', text)

def white\_space\_fix(text):

return ' '.join(text.split())

def remove\_punc(text):

exclude = set(string.punctuation)

return ''.join(ch for ch in text if ch not in exclude)

def lower(text):

return text.lower()

return white\_space\_fix(remove\_articles(remove\_punc(lower(s))))

def get\_tokens(s):

if not s: return []

return normalize\_answer(s).split()

def compute\_exact(a\_gold, a\_pred):

return int(normalize\_answer(a\_gold) == normalize\_answer(a\_pred))

def compute\_f1(a\_gold, a\_pred):

gold\_toks = get\_tokens(a\_gold)

pred\_toks = get\_tokens(a\_pred)

common = collections.Counter(gold\_toks) & collections.Counter(pred\_toks)

num\_same = sum(common.values())

if len(gold\_toks) == 0 or len(pred\_toks) == 0:

# If either is no-answer, then F1 is 1 if they agree, 0 otherwise

return int(gold\_toks == pred\_toks)

if num\_same == 0:

return 0

precision = 1.0 \* num\_same / len(pred\_toks)

recall = 1.0 \* num\_same / len(gold\_toks)

f1 = (2 \* precision \* recall) / (precision + recall)

return f1

def get\_raw\_scores(dataset, preds):

exact\_scores = {}

f1\_scores = {}

for article in dataset:

for p in article['paragraphs']:

for qa in p['qas']:

qid = qa['id']

gold\_answers = [a['text'] for a in qa['answers']

if normalize\_answer(a['text'])]

if not gold\_answers:

# For unanswerable questions, only correct answer is empty string

gold\_answers = ['']

if qid not in preds:

print('Missing prediction for %s' % qid)

continue

a\_pred = preds[qid]

# Take max over all gold answers

exact\_scores[qid] = max(compute\_exact(a, a\_pred) for a in gold\_answers)

f1\_scores[qid] = max(compute\_f1(a, a\_pred) for a in gold\_answers)

return exact\_scores, f1\_scores

def apply\_no\_ans\_threshold(scores, na\_probs, qid\_to\_has\_ans, na\_prob\_thresh):

new\_scores = {}

for qid, s in scores.items():

pred\_na = na\_probs[qid] > na\_prob\_thresh

if pred\_na:

new\_scores[qid] = float(not qid\_to\_has\_ans[qid])

else:

new\_scores[qid] = s

return new\_scores

def make\_eval\_dict(exact\_scores, f1\_scores, qid\_list=None):

if not qid\_list:

total = len(exact\_scores)

return collections.OrderedDict([

('exact', 100.0 \* sum(exact\_scores.values()) / total),

('f1', 100.0 \* sum(f1\_scores.values()) / total),

('total', total),

])

else:

total = len(qid\_list)

return collections.OrderedDict([

('exact', 100.0 \* sum(exact\_scores[k] for k in qid\_list) / total),

('f1', 100.0 \* sum(f1\_scores[k] for k in qid\_list) / total),

('total', total),

])

def merge\_eval(main\_eval, new\_eval, prefix):

for k in new\_eval:

main\_eval['%s\_%s' % (prefix, k)] = new\_eval[k]

def plot\_pr\_curve(precisions, recalls, out\_image, title):

plt.step(recalls, precisions, color='b', alpha=0.2, where='post')

plt.fill\_between(recalls, precisions, step='post', alpha=0.2, color='b')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.xlim([0.0, 1.05])

plt.ylim([0.0, 1.05])

plt.title(title)

plt.savefig(out\_image)

plt.clf()

def make\_precision\_recall\_eval(scores, na\_probs, num\_true\_pos, qid\_to\_has\_ans,

out\_image=None, title=None):

qid\_list = sorted(na\_probs, key=lambda k: na\_probs[k])

true\_pos = 0.0

cur\_p = 1.0

cur\_r = 0.0

precisions = [1.0]

recalls = [0.0]

avg\_prec = 0.0

for i, qid in enumerate(qid\_list):

if qid\_to\_has\_ans[qid]:

true\_pos += scores[qid]

cur\_p = true\_pos / float(i+1)

cur\_r = true\_pos / float(num\_true\_pos)

if i == len(qid\_list) - 1 or na\_probs[qid] != na\_probs[qid\_list[i+1]]:

# i.e., if we can put a threshold after this point

avg\_prec += cur\_p \* (cur\_r - recalls[-1])

precisions.append(cur\_p)

recalls.append(cur\_r)

if out\_image:

plot\_pr\_curve(precisions, recalls, out\_image, title)

return {'ap': 100.0 \* avg\_prec}

def run\_precision\_recall\_analysis(main\_eval, exact\_raw, f1\_raw, na\_probs,

qid\_to\_has\_ans, out\_image\_dir):

if out\_image\_dir and not os.path.exists(out\_image\_dir):

os.makedirs(out\_image\_dir)

num\_true\_pos = sum(1 for v in qid\_to\_has\_ans.values() if v)

if num\_true\_pos == 0:

return

pr\_exact = make\_precision\_recall\_eval(

exact\_raw, na\_probs, num\_true\_pos, qid\_to\_has\_ans,

out\_image=os.path.join(out\_image\_dir, 'pr\_exact.png'),

title='Precision-Recall curve for Exact Match score')

pr\_f1 = make\_precision\_recall\_eval(

f1\_raw, na\_probs, num\_true\_pos, qid\_to\_has\_ans,

out\_image=os.path.join(out\_image\_dir, 'pr\_f1.png'),

title='Precision-Recall curve for F1 score')

oracle\_scores = {k: float(v) for k, v in qid\_to\_has\_ans.items()}

pr\_oracle = make\_precision\_recall\_eval(

oracle\_scores, na\_probs, num\_true\_pos, qid\_to\_has\_ans,

out\_image=os.path.join(out\_image\_dir, 'pr\_oracle.png'),

title='Oracle Precision-Recall curve (binary task of HasAns vs. NoAns)')

merge\_eval(main\_eval, pr\_exact, 'pr\_exact')

merge\_eval(main\_eval, pr\_f1, 'pr\_f1')

merge\_eval(main\_eval, pr\_oracle, 'pr\_oracle')

def histogram\_na\_prob(na\_probs, qid\_list, image\_dir, name):

if not qid\_list:

return

x = [na\_probs[k] for k in qid\_list]

weights = np.ones\_like(x) / float(len(x))

plt.hist(x, weights=weights, bins=20, range=(0.0, 1.0))

plt.xlabel('Model probability of no-answer')

plt.ylabel('Proportion of dataset')

plt.title('Histogram of no-answer probability: %s' % name)

plt.savefig(os.path.join(image\_dir, 'na\_prob\_hist\_%s.png' % name))

plt.clf()

def find\_best\_thresh(preds, scores, na\_probs, qid\_to\_has\_ans):

num\_no\_ans = sum(1 for k in qid\_to\_has\_ans if not qid\_to\_has\_ans[k])

cur\_score = num\_no\_ans

best\_score = cur\_score

best\_thresh = 0.0

qid\_list = sorted(na\_probs, key=lambda k: na\_probs[k])

for i, qid in enumerate(qid\_list):

if qid not in scores: continue

if qid\_to\_has\_ans[qid]:

diff = scores[qid]

else:

if preds[qid]:

diff = -1

else:

diff = 0

cur\_score += diff

if cur\_score > best\_score:

best\_score = cur\_score

best\_thresh = na\_probs[qid]

return 100.0 \* best\_score / len(scores), best\_thresh

def find\_all\_best\_thresh(main\_eval, preds, exact\_raw, f1\_raw, na\_probs, qid\_to\_has\_ans):

best\_exact, exact\_thresh = find\_best\_thresh(preds, exact\_raw, na\_probs, qid\_to\_has\_ans)

best\_f1, f1\_thresh = find\_best\_thresh(preds, f1\_raw, na\_probs, qid\_to\_has\_ans)

main\_eval['best\_exact'] = best\_exact

main\_eval['best\_exact\_thresh'] = exact\_thresh

main\_eval['best\_f1'] = best\_f1

main\_eval['best\_f1\_thresh'] = f1\_thresh

def main():

with open(OPTS.data\_file) as f:

dataset\_json = json.load(f)

dataset = dataset\_json['data']

with open(OPTS.pred\_file) as f:

preds = json.load(f)

if OPTS.na\_prob\_file:

with open(OPTS.na\_prob\_file) as f:

na\_probs = json.load(f)

else:

na\_probs = {k: 0.0 for k in preds}

qid\_to\_has\_ans = make\_qid\_to\_has\_ans(dataset) # maps qid to True/False

has\_ans\_qids = [k for k, v in qid\_to\_has\_ans.items() if v]

no\_ans\_qids = [k for k, v in qid\_to\_has\_ans.items() if not v]

exact\_raw, f1\_raw = get\_raw\_scores(dataset, preds)

exact\_thresh = apply\_no\_ans\_threshold(exact\_raw, na\_probs, qid\_to\_has\_ans,

OPTS.na\_prob\_thresh)

f1\_thresh = apply\_no\_ans\_threshold(f1\_raw, na\_probs, qid\_to\_has\_ans,

OPTS.na\_prob\_thresh)

out\_eval = make\_eval\_dict(exact\_thresh, f1\_thresh)

if has\_ans\_qids:

has\_ans\_eval = make\_eval\_dict(exact\_thresh, f1\_thresh, qid\_list=has\_ans\_qids)

merge\_eval(out\_eval, has\_ans\_eval, 'HasAns')

if no\_ans\_qids:

no\_ans\_eval = make\_eval\_dict(exact\_thresh, f1\_thresh, qid\_list=no\_ans\_qids)

merge\_eval(out\_eval, no\_ans\_eval, 'NoAns')

if OPTS.na\_prob\_file:

find\_all\_best\_thresh(out\_eval, preds, exact\_raw, f1\_raw, na\_probs, qid\_to\_has\_ans)

if OPTS.na\_prob\_file and OPTS.out\_image\_dir:

run\_precision\_recall\_analysis(out\_eval, exact\_raw, f1\_raw, na\_probs,

qid\_to\_has\_ans, OPTS.out\_image\_dir)

histogram\_na\_prob(na\_probs, has\_ans\_qids, OPTS.out\_image\_dir, 'hasAns')

histogram\_na\_prob(na\_probs, no\_ans\_qids, OPTS.out\_image\_dir, 'noAns')

if OPTS.out\_file:

with open(OPTS.out\_file, 'w') as f:

json.dump(out\_eval, f)

else:

print(json.dumps(out\_eval, indent=2))

if \_\_name\_\_ == '\_\_main\_\_':

OPTS = parse\_args()

if OPTS.out\_image\_dir:

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

main()